A COMPOSING COMPONENTS

--> react is tree of component so we can compose components together to build complex ui
--> exp: dari counter yg kita buat sebelumnya, kita buat list of counter
--> Langkah 1 atur Counters:
1) buat file baru yang berisi list of counter : nama file counter.jsx:
2) di index.js ganti yg sebelumnya pusatnya di `<Counter />` menjadi di `<Counters />` :

```
ReactDOM.render(<Counters />, document.getElementById('root'));
```

3) import juga Counters di index.js:

```
import Counters from './components/counters';
```

--> Langkah 2 masukin Counter ke Counters:
1) import Counter: `import Counter from './counter';`
2) masukin Counter dalam punya render berapapun :

```
<div>
   <Counter />
   <Counter />
   <Counter />
</div>
```

--> Langkah 2 advanced:
--> daripada ngasih manual, kita bisa buat tiap <Counter /> mempunyai id, caranya buat array counters yang berisi id2:
1) buat array:

```
counters : [
  {id : 1},
  {id : 2},
  {id : 3}
]
```

2) map array counters dan kasih key:

```
{this.state.counters.map( (a) => <Counter key={a.id} />)}
```

B PASSING DATA TO COMPONENTS

--> ide : tiap anggota array counters di class Counters selain dikasih id, dikasih 'isi', 'isi' ini berisi state.count pya class Counter,jadi ketika diklik, 'isi' juga akan bertambah, dan value dari count juga mengikuti value dari ‘isi’ gimana passing datanya?
--> jawab : pake props
--> Different between state and props:
a) Props :
- Immutable
- to pass data & event handlers down to your child components.
- have better performance

b) State:
- Mutable
- to store the data your current page needs in your controller-view.
- has worse performance
- should not be accessed from child components

--> Penggunan disini:
1) tambaha 'isi' di array counters :

```
state = {
  counters : [
    {id : 1, isi : 0},
    {id : 2, isi : 0},
    {id : 3, isi : 0}
  ]
}
```

2) taro 'isi' ke props:

```
{this.state.counters.map( (a) => <Counter key={a.id} isi = {a.isi} />)}
```

//syataxnya gini : nama props = blabla

--> disini "isi" udah bisa dioper dan ada di this.props
3) ubah state.count di class Counter agar nilainya mengikuti 'isi' :

```
state = {
  count : this.props.isi,
};
```

C PASSING CHILDREN

D) DEBUGGING REACT APP

E) PROPS VS STATE

F) RAISING AND HANDLING EVENT

--> kasus: pengen hapus salah satu list
--> problem : daftar list ada di Counters sedangkan tombol delete ada di Counter, gimana cara hapusnya?
padahal ada kaedah di react :
{the component that owns a piece of the state, should be the one modifying it}
--> solusi : pake cara raise and handle ; bwt fungsi di class Counter buat manggil fungsi di class Counters yang fungsi ini bertugas ngehapus list di this.state.counters.
--> Langkah :
1) di Class Counter : buat buttton yg berisi fungsi untuk manggil fungsi di class Counters:

```
<button onClick={this.props.onDelete} className = "btn btn-danger btn-sm m-2">Delete</button>
```

2) buat fungsi buat hapus anggota array counters dan pass pake props ke class Counter

```
handleDelete = () => {
    console.log("handleDelete");
};
```

//dalam render :

```
<Counter key={a.id} isi = {a.isi} onDelete= {this.handleDelete}>
```

--> Sementara belum sampai hapus list, cuma manggil fungsinya dulu

G) UPDATING STATE

--> problem : state kan immutable, berati kan nggk bisa diupdate? solusi : bisa kalo ditimpa
--> gimana cara nimpa klo pengen hapus salah satu list counter? pass idnya pake argument dalam fungsi, setelah dapat id, timpa pake method array.filter(cuma id yang selain di click yang ditampilin)
--> Langkah2 :
1) pass id dari Counters ke Counter:

```
<Counter key={a.id} isi = {a.isi} onDelete= {this.handleDelete} id = {a.id}>
```

2) di Counter, masukin id untuk argument :

```
<button onClick={() => this.props.onDelete(this.props.id)} className = "btn btn-danger btn-sm m-2">Delete</button>
```

3) di Counters, filter dan timpa counters:

```
handleDelete = (idCounter) => {
  const a = this.state.counters.filter((oneCounter) => oneCounter.id !== idCounter);
          //akan menfilter, cuma id yang bukan sesuai argument yang dimasukin ke const a
  this.setState({counters : a});
};
```

--> Note : jika nama const a (menjadi const counter) dan counters sama, setState() bisa disingkat menjadi: `this.setState({counters});`

H) SINGLE SOURCE OF TRUTH

--> ternyata ada masalah di codingan yang dibuat : terdapat 2 source state yang sama disini, yaitu counters.isi milik Counters dan this.state.count milik Counter. Meskipun this.state.count merujuk ke this.props.isi, ketika counters.isi berubah, this.props.isi akan ikut berubah, tetapi this.state.count kadang berubah kadang nggk
--> Kalo kita mengubah counters.isi di Counters, this.state.count nggk akan berubah, karena tiap class memiliki local state masing2.
--> contoh : ketika kita mereset semua angka anggota menjadi 0:

```
handleReset = () => {
    const a = this.state.counters.map(allCounter => {
    allCounter.isi = 0;
    return allCounter
  });
  this.setState({counters : a });
};
```

//di counters berubah tapi di count nggk
--> Solusi? buat single source of truth. this.state.count(local state punya Counter) dibuang , dan buat semua merujuk ke counters(local state punya Counters)
--> Intinya : jangan sampai ada 2 state di class berbeda yg memiliki nilai sama, jika ada, satuin di parent, yang di child dihapus

I) REMOVING THE LOCAL STATE

--> Kita bakal menghapus this.state.count dan data akan berasal dari this.props.isi aja. Class Counter akan menjadi controlled component yang menerima data dan raise function ke parent

--> Langkah2:

1. hapus this.state.count
2. pindah dan sesuaikan fungsi yang berhubungan dengan this.state.count ke parent(Counters)

1) oper this.state.counters, handleIncrement ke Counter :

```
{this.state.counters.map( (a) => (
      <Counter
        key={a.id}
        isi = {a.isi}
        onDelete= {this.handleDelete}
        onIncrement = {this.handleIncrement}
        id = {a.id}>
        counter = {a}   //disini
      </Counter>
))}
```

2) fungsi naik():

--> ubah namanya menjadi : handleIncrement(nama aja si) :ubah syantax menjadi :

```
handleIncrement = (a) => {
      //parameter "a" itu berisi anggota array counters(bukan array counters)
  const index = this.state.counters.indexOf(a);
  const ygDiubah = this.state.counters[index].isi++;
  this.setState({ygDiubah});
}
```

//bisa gini tapi bakal ada error : Line 48: Do not mutate state directly. Use setState() react/no-direct-mutation-state , gara ngubah state langsung

//solusi:

```
handleIncrement = (a) => {
   const allCounter = [...this.state.counters]; //clone semua anggota counters
   const index = allCounter.indexOf(a); //nyari a itu ada di index mana dalam allCounter
   allCounter[index] = {...a}; //ngambil id dan isi counter, karene dia object makanya {}
   allCounter[index].isi++;
   this.setState({counters : allCounter});
}
```

--> raise naik(yang diubah nama variablenya buat ngoper menjadi onIncrement) dari Counters:

```
<button onClick={() =>this.props.onIncrement(this.props.counter)} className="btn btn-
 secondary btn-sm">Increment</button>
```

//disini yg digunain parameter buat handleIncrement adalah this.props.counter yang berisi anggota array counters, tinggal gimana handleIncrement ngolahnya

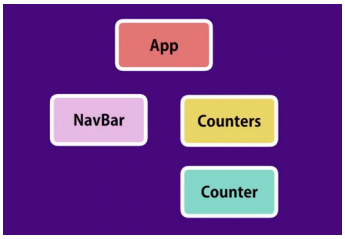
3) di ubahWarnaLencana() : ganti reference ke anggota counters dengan id ttn :

```
ubahWarnaLencana(){
  let warna = "badge m-2 badge-";
  warna += this.props.counter.isi === 0 ? "warning" : "primary" ; //disini
  return warna;
}
```

4) di formatCount() : ganti referencce ke anggota counters :

```
formatCount(){
   const {isi} = this.props.counter;
   return isi === 0 ? 'Zero' : isi;
}
```

J) MULTIPLE COMPONENT IN SYNC



--> Nambahin navbar :

1) Karena parent sekarang bukan Counters tapi di App, di index.js diubah DOM utamanya:

```
ReactDOM.render(<App />, document.getElementById('root'));
```

2) buat navbar.jsx, isi dengan :

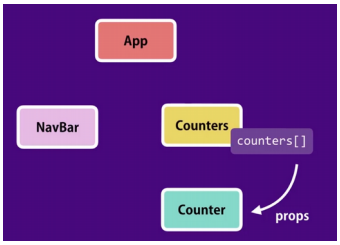
```
<nav className="navbar navbar-light bg-light">
   <a className="navbar-brand" href="#">
        Navbar
   </a>
</nav>
```

3) di App : panggil Navbar dan Counters : import juga:

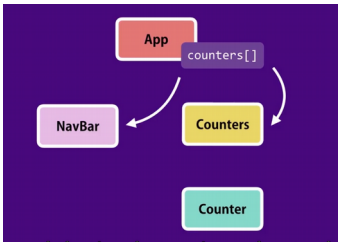
```
<React.Fragment>
   <Navbar />
      <main className="container">
        <Counters />
   </main>
</React.Fragment>
```

--> Sekarang, kalo pengen di samping navbar ada badge yang ngasih tau total angka yang ada, gimana caranya? Kan nggk ada hnbungan parent-child antara Navbar dan Counters?

--> Cara : U must lift the state up



awalnya gini



dilift up counters[] nya

K) LIFTING STATE UP

--> Pindahin counters[] dan semua yang mengikutinya dari Counters ke App:

1) counters[], handleIncrement, handleReset, handleDelete pingah ke App

2) buat props dari App ke Counters:

```
<Counters
  counters = {this.state.counters}
  onReset = {this.handleReset}
  onIncrement = {this.handleIncrement}
  onDelete = {this.handleDelete}
/>
```

3) Sesuaikan Counters:

```
<button
   onClick={this.props.onReset} //ini
   className = "btn btn-primary btn-sm m-2">Reset</button>
{this.props.counters.map( (a) => (
   <Counter
      key={a.id}
      isi = {a.isi}
      onDelete= {this.props.onDelete} //ini
      onIncrement = {this.props.onIncrement} //ini
      id = {a.id}
      counter = {a}>
   </Counter>
))}
```

--> Cara buat di samping navbar ada badge yang ngasih tau list array yang nggk kosong :

1) di App : bikin props untuk Navbar yang berisi jumlah anggota yang nilai isi tidak kosong:

```
<Navbar totalNavbar={this.state.counters.filter(c => c.isi > 0).length}/>
```

2) Tampilin di Navbar:

```
<span className="badge badge-pill badge-secondary">{this.props.totalNavbar}</span>
```

L) STATELESS FUNCTIONAL COMPONENT

--> ini bisa digunakan ketika nggk ada state di fungsi kita

-->Syntax: mirip dengan pure react component,bedaya pake fungsi, props dioper diargumen, render nggk ada, selainnya sama :

```
import React, {Component} from 'react';

const Navbar = (props) => {    //disini pake fungsi dan ngoper props
  return (      //disini render nggk ada
    <nav className="navbar navbar-light bg-light">
      <a className="navbar-brand" href="#">
        Navbar
        <span className="badge badge-pill badge-secondary">{props.totalNavbar}</span>
      </a>                                                    //disini this hilang
    </nav>
  );
}

export default Navbar;
```

M) DESTRUCTURING ARGUMENTS

1) di counters:di render:

```
const {onReset,counters, onDelete,onIncrement} = this.props;
```

```
onClick={this.props.onReset}                menjadi --> onClick={onReset}
{this.props.counters.map( (a) => (          menjadi --> {counters.map( (a) => (
onDelete= {this.props.onDelete}             menjadi --> onDelete= {onDelete}
onIncrement = {this.props.onIncrement}      menjadi --> onIncrement = {onIncrement}
```

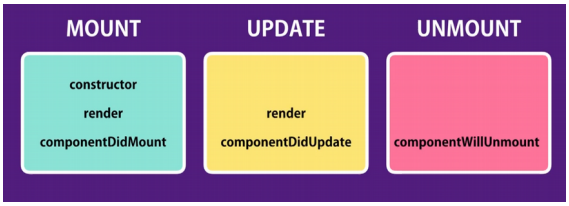
2) di Navbar:

```
const Navbar = (props) => {                 menjadi --> const Navbar = ({totalNavbar}) => {

{props.totalNavbar}                         menjadi --> {totalNavbar}
```

N) LIFECYCLE HOOK

--> Aplikasi react berjalan dalam beberapa fase, yang paling terjadi terdapat 3 fase:



1) Fase Mount --> terjadi ketika instance dari component kita dibuat dan dimasukkan ke DOM
2) Fase Render --> terjadi ketika props dan state berubah
3) Fase Unmount --> terjadi ketika component dihapus dari DOM,

--> Dalam 3 fase tersebut, ada beberapa method spesial yang bisa kita tambahkan dan react otomatis memanggil method ini --> method ini disebut lifecycle hook

--> lifecycle hook ngijinin kita mengaitkan (hook) ke moment tertentu selama alur hidup component (lifecycle of component) dan melakukan sesuatu.

--> Method2 special: terjadi secara berurutan :

1) Fase Mount --> `constructor()`, `render()`, `componentDidMount()`
2) Fase Render --> `render()`, `componentDidUpdate()`
3) Fase Unmount --> `componentWillUnmount()`

O) MOUNTING PHASE

1) Constructor()

--> Constructor is called only once when an instance of a class is created

--> Bukti :

```
constructor(){
   super();
   console.log("App - Constructor");
}
```

--> Muncul kata : App - Constructor 1 kali pada console

--> Ini bisa digunain buat inisialisi state secara langsung (tanpa setState()) karena setState() berjalan ketika proses render()

```
constructor(props){
  super(props);
  console.log("App - Constructor");
  this.state = this.props.something;
}
```

2) render() --> kaya biasa yang kita pake, buat ngerender virtual DOM ke DOM asli

--> Ketika parent di render, bocil2nya ikul ke render

3) componentDidMount()

--> dipangggil setelah component sudah direnderer ke DOM asli

--> Waktu yang sempurna buat membuat AJAX call to get data from the server

--> Aplikasi : panggil, lalu setState ke terbaru

```
componentDidMount(){
    //AJAX call
    this.setState({isi_AJAX});
}
```

--> Kalo kita console.log,urutan yang dipanggil akan seperti ini :

App - Constructor
App - Rendered
App - Mounted

P) UPDATING PHASE

--> Terjadi jika state atau props berubah

--> Ketika parent di render, bocil2nya ikul ke render

--> Contoh : ketika handleIncrement di klik, .isi bertambah , parent(App) dan bocil2nya akan ke render juga :

App - Rendered
Navbar - Rendered
Counters - Rendered
Counters - Rendered

--> When entire component tree is rendered that doesnt mean that entire data is updated,when component is rendered we basically get a react element, so that is updating our virtual DOM. React will then look at the virtual DOM, it also have a copy of the old virtual DOM. That's why we shouldnt update directly so we can have 2 object berbeda yang merujuk ke memory. Kita punya Old virtual DOM dan New virtual DOM. Then react will figure out what is changed and based on that, it will update the real DOM accordingly

--> Fungsi:

1) componentDidUpdate()

--> Guna : Method ini dipanggil setelah component di update, yang berarti kita punya state/props baru yang bisa kita gunakan untuk membnadingkan dengan state/props lama. Jika da perubahan, kita bisa buat AJAX request untuk mengambil data baru dari server

--> Contoh : di Counter :

```
componentDidUpdate(prevProps, prevState){
  console.log("prevProps", prevProps);
  console.log("prevState", prevState);
  if (prevProps.counter.isi !== this.props.counter.isi){
    //Ajax call and get new data from the server
  }
}
```

Q) UNMOUNTING PHASE

1) componentWillUnmount() :

--> Fungsi dipanggil setelah component dihilangkan dari DOM

--> exp : di Counter :

```
componentWillUnmount(){
  console.log("Counter - Unmount");
}
```

--> Hasil :

App - Rendered
Navbar - Rendered
Counters - Rendered
3Counters - Rendered
Counter - Unmount